Mar 13, 2025

Semantic Similarity - Reports

Authors: Tamir Nizri , Eyal Segev, Lior Hagay

Design

App.java:

Orchestrates an Amazon EMR MapReduce job flow composed of four sequential tasks executed via custom JAR files stored on AWS S3. It sets up the EMR cluster environment, submits the tasks, and manages their lifecycle through the AWS Java SDK.

Step1.java: Count(F is f) Calculation

Purpose:

This step is designed to compute the frequency of each feature across the entire corpus.

- **Mapper**: Reads each line (headword and its features) and emits each observed feature along with its associated headword and its count as a key-value pair, where the key is the feature and the value is the headword+count. Emits (feature, headword + count).
  In this Step we assure that the headwords we are analyzing are only headwords in the golden-standard dataset.
- **Reducer**: Aggregates these pairs to calculate the total number of occurrences of each feature across the corpus (count(F is f)). This data is crucial for calculating the formulas of the co-occurence vectors. Emits (headword , features list where one of them has its count(F is f) + general count).

Input/Output:

- **Input**: A set of text files from the English All - Biarcs dataset of Google Syntactic N-Grams

- **Output**: Each line of output specifies a headword followed by its associated features and the general count, where one of the features has its count(F is f).

Step2.java: Aggregation of features with count_f_is_F

Purpose:

Attach to each headword all the features that have a count_f_is_F calculated already from Step1.

- **Mapper**: Extracts features within the context of their sentences, using a key of the headword and the sentence. This helps maintain the feature-sentence relationship. Emits (headword + sentence, a list of features where only one feature has count_F_is_f)
- **Reducer**: Reorganizes the data for a specific headword+sentence to aggregate all features associated with each headword in the context of a sentence, along with their count_f_is_F (the general feature count for a specific feature in the corpus). Emits (headword, a list of all the features with their count_f_is_F).

Input/Output:

- **Input**: Key-Value pairs of the previous step.
- **Output**: Each line of output specifies a headword followed by its associated features and the general count. (This time all the features are with count_f_is_F).

Step 3: Count(f,l), Count(F), Count(L) Calculation

Purpose:

To compute more corpus statistics that are crucial for understanding the distribution and frequency of features and headwords.

- **Mapper**: Calculates the total counts of features (count F) and headwords (count L) across the entire corpus and stores them in Hadoop global counters. Emits(headword, list of features with their count(F is f) + general count|).
- **Reducer**: Computes the frequency of feature co-occurrences with specific headwords (count(f , l)). Emits(headword, list of features with their count(F is f) and their count(f, l)).

Input/Output:

- **Input**: Key-Value pairs of the previous step.
- **Output**: Headwords with their list of features, where every feature has every statistic needed for computing the formulas of co-occurrence vectors.

Step 4: Calculating Semantic Similarity Scores Between Headword Pairs

Purpose:

To produce pairs of words according to the word-relatedness.txt file and combine their vectors calculated in previous steps in order to compute final semantic similarity scores between these headwords based on their feature vectors. The Mapper makes for each feature 4 values of measurement and prepares the pair for the reducer to calculate the distances using 6 different calculation methods.

- **Mapper**: Takes the vectors from the previous steps of specific headwords that are filtered to the words in word-relatedness.txt file. Finds words that are paired with each headword that we got as input and prepares the values of each vector we got in 4 measurement methods as specified in the task.
  Emits (pair of headwords (ordered lex), vector of one headword with 4 measurement methods)
- **Reducer**: Processes these vectors to calculate the distance between them for a specific pair using 6 distance calculation methods. Resulting in a 24 long vector for each pair.
  Emits (pair of headwords, vector of 24 numbers for each measurement method and distance method)

Input/Output:

- **Input**: The vectors we got from Step 3.
- **Output**: Pairs of words and their 24 long vector.

Helper Classes:

Stemmer: The Stemmer we got from the assignment

ConvertToARFF: Converts the outputs from the Map-Reduce Job Flow to ARFF format for WEKA program.

WekaClassification: Runs the WEKA program with the ARFF file and writes the statistical info from the WEKA results.

Communication and Memory usage - Asymptotic

Let n be the number of different words in the golden-standard dataset.
**We are analyzing according to the number of words in the golden-standard dataset because after the Mapper of Step1 we filtered already all the headwords that appear in that dataset.

Step1:

- Number of key-value pairs from Mapper to Reducer: approximately O(n). This is because in the corpus each sentence has approximately 2-3 features and we are emitting each feature of a headword as a key resulting in 3n = O(n) keys. As we explained before we are sending to the reducer only the features related to headwords contained in the gold-standard dataset.
- Memory assumption: The amount of words in the Gold-standard Dataset.

Step2:

- Number of key-value pairs from Mapper to Reducer: O(n)
- Memory assumption: O(1) - For each headword+sentence (the key) we have a hashset of their features - approximately 3 features, resulting in O(1) memory.

Step3:

- Number of key-value pairs from mapper to reducer: O(n)
- Memory assumption: all of the features in the corpus for a single headword.

Step4:

- Number of key-value pairs from mapper to reducer: O(n^2) - number of pairs in the golden-standard dataset.
- Memory assumption: For each word we are saving in a HashMap all the words that can appear next to it in the golden-standard dataset + All of the features in the corpus for a specific pair in the golden-standard dataset.

WEKA and Classification

Precision - ratio of correctly classified positive cases out of all cases predicted as positive.

Recall - ratio of correctly classified positive cases out of all actual positive cases.

F1 - balancing Precision and Recall, evaluating overall model performance.

** As described in the assignment we were told to choose any classification algorithm we wanted, Our choice was to use the Decision Tree (J48) Algorithm

<u>10 files:</u>

Decision Tree (J48) Algorithm

- **Precision:** 0.920
- **Recall:** 0.946
- **F1:** 0.933

<u>100 files:</u>

Decision Tree (J48) Algorithm

- **Precision:** 0.932
- **Recall:** 0.950
- **F1:** 0.940

Communication and Memory usage - Details from AWS

Component	Memory usage 10%	Memory usage 100%	Key-values from mapper to reducer - 10%	Key-Values from mapper to reducer - 100%
Step 1	369 GB	3352.76 GB	212,076,082 pairs, approximately 14.27 GB	2,120,390,400 pairs, approximately 141.64 GB
Step 2	159 GB	1541 GB	227,132,225 pairs, approximately 14.57 GB	2,171,482,520 pairs, approximately 143.41 GB
Step 3	65.76 GB	628.15 GB	53,527,701 pairs, approximately 3.28 GB	514,174,300 pairs, approximately 30.08 GB
Step 4	16.34 GB	161.22 GB	287 pairs, approximately 80.61 MB	2694 pairs, approximately 786 MB